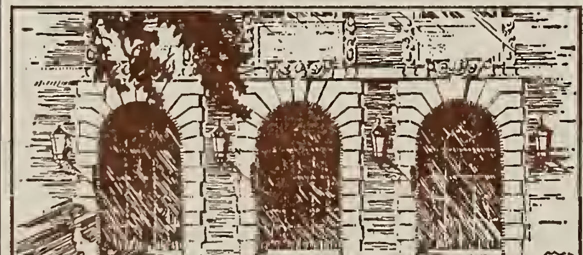


[redacted]

510.84
Il6r
no.578
cop 2

Math

UIUCDCS-R-73-578

SIMULATION ANALYSIS OF A NETWORK COMPUTER

by

Fredrick R. Salz

May 18, 1973

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

UIUCDCS-R-73-578

SIMULATION ANALYSIS OF A NETWORK COMPUTER

by

FREDRICK R. SALZ

May 18, 1973

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

*Submitted in partial fulfillment of the requirements for the Master of Science degree in Computer Science in the Graduate College of the University of Illinois at Urbana-Champaign and supported in part by the National Science Foundation under Grant No. GJ 28289.

ACKNOWLEDGMENT

I would like to sincerely thank Professor E. K. Bowdon, Sr. of the Department of Computer Science for his encouragement and guidance as an adviser and friend. I appreciate his efforts that made this work possible.

I would also like to thank Mr. R. O. Skinner of the Computing Services Office for all of his time in supplying information on the operation of the IBM 360/75 necessary for developing the simulation model. Finally, my thanks to Mrs. Gayanne Carpenter whose assistance and superb typing were invaluable in the preparation of this paper.

This research was supported in part by the National Science Foundation under Grant No. GJ 28289.

TABLE OF CONTENTS

1. INTRODUCTION

Before it is possible to evaluate the performance of a computing system, the goals that the system is to achieve must be defined. These goals, however, are not the same for both the user and the computer center manager. The overall performance of the system is, therefore, some combination of these goals.

To the individual user, the system is at its best when the "turnaround time" for his job (the elapsed time between submitting his cards and receiving his printed output) is at a minimum. He is not really concerned with what happens to the other users of the system as long as he can obtain the desired results within as little time as possible. In some cases, the turnaround time is satisfactory if the results are available before a certain specified deadline.

On the other hand, the computer center manager sees the total system throughput as the important criterion for evaluating the performance of the system. Usually, more jobs being processed per hour results in more revenue being collected by the system. From a management point of view, the income per hour is of primary concern. In addition, the management strives to give satisfactory service to all of the users of the facility. Unlike the individual's self-centered attitude, the manager would like to improve the turnaround time for as many users as possible.

A third point of view could be that of the system itself. The criteria involved here would include obtaining maximum use of resources. There is a substantial cost involved with owning (or renting) and operating computer machinery, and if more use could be made of the existing facilities, perhaps additional equipment expenditures would not be necessary.

This paper presents the results of the analysis of scheduling algorithms for a network computer based on the desire to fulfill these goals.

2. BASIC CONCEPTS

2.1. Use of Simulation

Before new proposals are implemented on a computer system, it is desirable to study the effect that these changes will have on the performance of the system. Many ideas that seem favorable in discussions and on paper may not yield any real improvements, and in fact they may even result in a serious degredation of performance. A number of techniques can be employed to give the designers an opportunity to study their proposals. One method is to write the new feature into the present system, and at times when the facilities are unavailable to the general populace, run the system with the new features. Obviously, for a complex change, this would be extremely costly in both the programmer's time, and unproductive machine time. Unproductive, that is, from the user's point of view (since he cannot run any of his programs), and from the center manager's point of view (since revenue is lost). However, this would allow the maximum amount of performance data to be collected since all data normally available to the system would be available during the test runs.

A second alternative, developing analytical models based on queueing theory analysis, has certain advantages over the previous method: the major one being that no computer time is necessary for the analysis. Unfortunately, for a complex situation the model becomes extremely burdensome and a great number of simplifying assumptions must be made to make a solution possible. The information obtainable from the model (which is quite limited to begin with) tends to be in a form which is difficult to interpret. Analytical models on the other hand do have the advantage that the results are not influenced by statistical variation. For example, if an exponential distribution is assumed, and the equations are carried to a steady state solution, there is no dependence on the number of samples chosen or the time the system is allowed to run. However, by this stage, so many simplifying assumptions have been made that, even as good as the results are for the particular model, the model has strayed so far from the real-life conditions being studied that the results are nothing more than a fair approximation.

A third method, the one used for studying the algorithms presented in this paper, is to develop a simulation model that can operate under the current computer system. This procedure combines many of the advantages of the previous two, while minimizing the disadvantages. Although the computer is used to evaluate the proposals, the amount of computer time necessary is considerably less than the real time requirements of the first method.*

*The model used in this study required approximately a 1-to-3 ratio of 360/75 processor time to real time simulated. The actual ratio depends on the load of the system being studied.

Even though some simplifying assumptions must be made, they are not nearly as limiting as those necessary in analytical models. Not only can much more data be collected using the simulation model, but it is in a form more easily interpreted by the analyst. In fact, a complete simulation model will yield data not even available to the real system. For example, utilization data for the central processing unit, main memory, etc., is not usually maintained by the real system.

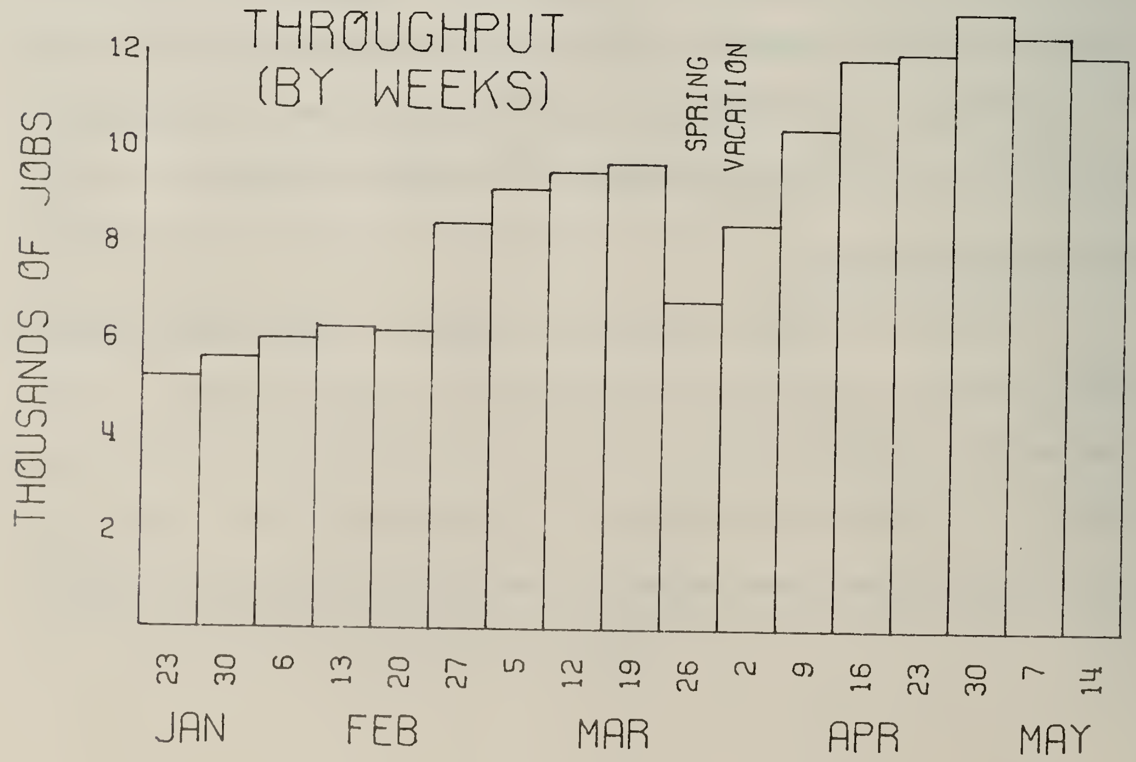
For the results obtained from a simulation model to be of value, they must be accurate both statistically and functionally. The simulated events must correspond to actual events, and the frequency and time between events must truly represent the real world. For this reason, the model used in this study was first developed to simulate the IBM 360/75 computing system currently in operation at the University of Illinois at Urbana-Champaign. Once the model accuracy was verified (as will be discussed in Chapter 3), the model could be used to evaluate new proposals with reasonable assurance of accurate results.

2.2. Network Concepts

Having developed a simulation model of a large scale computing system, attention was turned to the study of a network computer. Given n independent computer systems, with $n > 1$, the load L_i on system S_i will (usually) be different than the load L_j on system S_j if $j \neq i$. Therefore, if $L_i > L_j$, the idle resources of S_j could be utilized by S_i. This process of load leveling and resource sharing between centers is the basis for network computers.* For example, on a network such as ARPA [6], etc. there are many universities and research facilities sharing resources. The heaviest load at University computing centers occurs at the end of each semester when research projects are due, and for the rest of the year the load is relatively light.**

* In some cases, a job is transmitted from S_i to S_j because S_j offers some resource not available at S_i. This case is not considered in the discussion given here.

**At the University of Illinois at Urbana-Champaign, the system load for the Spring 1972 semester is shown below.



All universities, however, are not on the same schedule, with some semesters ending as much as two months apart. With these diverse schedules, the heavy load at one center will correspond to a light load at another. By transmitting a job from S_i to S_j where $L_i > L_j$ relieves some of the burden on S_i, while increasing resource utilization at S_j.

A general network of computing centers is shown in Figure 1. λ_i represents the arrival rate for jobs submitted at center i; μ_i is the service rate at center i; and α_{ij} is the rate at which jobs are transmitted from center i to center j.

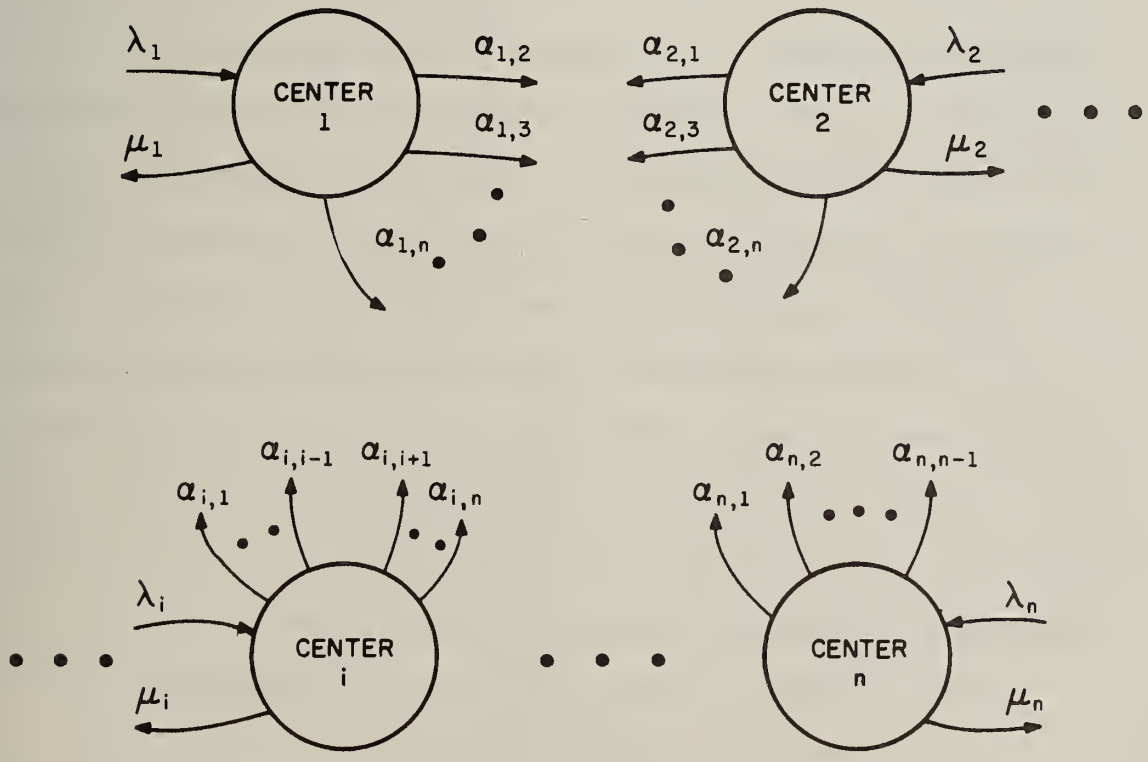


Figure 1. General Network Computer

2.3. Single Node Concepts

For each center, or node, of the network described above, scheduling algorithms were developed to facilitate the implementation of the networking concepts. In addition, new algorithms were designed to make greater use of the resources available.

The basis for the priority scheme developed for each center is the theory that maximum throughput is obtained by assigning the highest priority to jobs with the shortest processing time [8]. The time a job spends in the system (after execution has begun) is the sum of a number of factors including CPU processing time, time issuing input/output requests, time waiting for core, channel contention time, and ready wait time. The first three of these times can be approximated through user estimates, but the last two are a function of the load on the system at the time the jobs runs and cannot be determined in advance. Therefore, each job entering the system is assigned an initial static priority, PRT, based on the information available from the user.

The expected processing time based on user estimates, the static priority assigned, can be shown to be

$$\mathrm{PRT} = \mathrm{CPU} + \alpha\mathrm{IOR} + \beta\mathrm{COR}$$

where CPU = central processing unit processing time,

IOR = the number of input/output requests issued, and

COR = region of core requested (kilobytes).

α and β are multiplicative constants representing the processing time required for IOR and COR, respectively.* Jobs are ordered on the HASP queue according to their PRT, with the job with the smallest PRT being the first to be processed.

Once jobs are placed on the queue, information on system load can be dynamically used to reorder the waiting jobs so that their execution will increase the utilization of resources. The system can also guarantee a maximum waiting time for jobs by increasing a job's priority (decreasing its PRT) after a predetermined time has elapsed. A complete discussion of the static and dynamic priority assignment can be found in [3].

The PRT theories led directly to the study of two scheduling schemes that maximized the user's goals and the computer center manager's goals. Both schemes involved the use of rewards (monetary) given to the center by the user if exceptional service is rendered by the system. The first of these algorithms allows a user to specify a price he is willing to pay for improved service. This "Pay-for-priority" system, by assigning a higher initial priority (PRT) to a job willing to pay more for the same resources, allows a user to increase his turnaround time for selected jobs. At the same time, since the same resources are being used, with more revenue being collected, the center manager sees improvement in system performance.

*Through the use of system accounting records for previous years, α and β were determined to be 38 milliseconds and $\max(0, 7COR^2 - 1000\ COR)$ milliseconds, respectively. [5]

The second, and more elaborate algorithm, allows the user to specify rewards and deadlines as a basis for the price he will pay for service. The specifications basically consist of ordered pairs (r_i,d_i) where r_i is the reward the user is willing to pay to have his job finished by deadline d_i. Secondary and higher order deadlines are specified until the point where the job is considered worthless to the user, or the cost for running the job becomes greater than the amount of money that would be collected if the job were processed.

A <u>cost</u> <u>effective</u> system is realized by using the ratio or reward specified by the user to the cost of running his job as a basis for assigning priority to the job [1]. For example, suppose a programmer's meeting at which certain progress is to be reported is called for 8:00 Tuesday morning. His supervisor would like to see the results Monday afternoon to discuss what to say at the meeting. Under this system, the user could specify a high reward for obtaining results before Monday afternoon, a secondary reward for completion before Tuesday at 8:00, and some minimum payment to have it completed any time Tuesday or Wednesday. By meeting the primary deadline, the most revenue for the resources used will be collected by the system, and the user will have his results in the desired time.

The complete information vector necessary for a job T_i is then

$$(T_i, g/f, d_i, \tau_i, s_i)$$

where

T_i is the job name or other identifier,

g/f is the reward/cost ratio for meeting the task's deadline,

d_i is the task's deadline associated with g/f,

τ_i is the maximum processing time for this task, and

$s_i = d_i - \tau_i$ is the latest time at which the task may start processing in order to meet its deadline.

The contents of the information vector are used to determine the times at which jobs must be initiated so that they meet their deadlines. Reference [2] offers a complete description of the cost effective priority assignment. Since the individual centers are part of a network computer, if center i cannot meet this job's deadline and center j can, and the cost of transmission from center i to center j is less than the reward associated with meeting the deadline, the job can be transmitted to center j for processing.

3. CONSTRUCTION OF THE SIMULATOR

To evaluate the usefulness of the algorithms developed, they were implemented on a network simulator. The simulation model, written in IBM's GPSS (General Purpose Simulation System)[4] and run on an IBM 360/75 computer system, consists of three interconnected nodes as shown in Figure 2. Each node of the network consists of an independent computer system with independent job streams and service rates. "Communication lines" were established between nodes to allow for load leveling between centers.

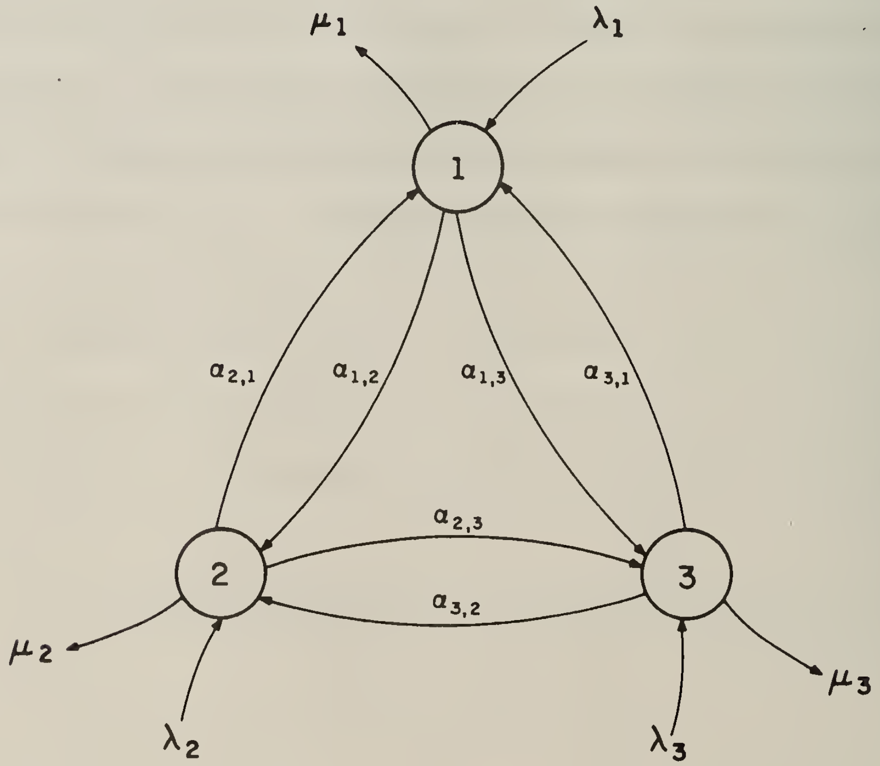


Figure 2. Three Node Network Computer

3.1. Discussion of Node

The nodes of the network were modelled after the IBM 360/75 under HASP and O.S. 360 as currently in operation at the University of Illinois at Urbana-Champaign. The various stages of execution a job follows as it goes through the system are described in the following paragraphs.

HASP Initiation

Figure 3 shows the logical structure of HASP (Houston Automatic Spooling Priority System) [9]. Jobs enter the system simultaneously from card readers, tapes, disks, or terminals. Upon arrival in the system, the job is placed on a HASP spool according to a priority determined by a number of factors including resource and reward estimates discussed earlier. The priorities are further divided into five priority classes (classes A,B,C,D, and X) based on the above factors and additional options specified by the user. Spools exist for each of the priority classes, and the job is assigned a position in the appropriate queue.

Seven HASP initiator/terminators select jobs from these spools when system resources become available. Parameters of the initiator determine which of the spools to search for the next job to be selected, and in what order. Once a job is selected for initiation, it is placed on the O.S. queue to be serviced by the operating system. The HASP initiator is put in a dormant state awaiting the job's completion.

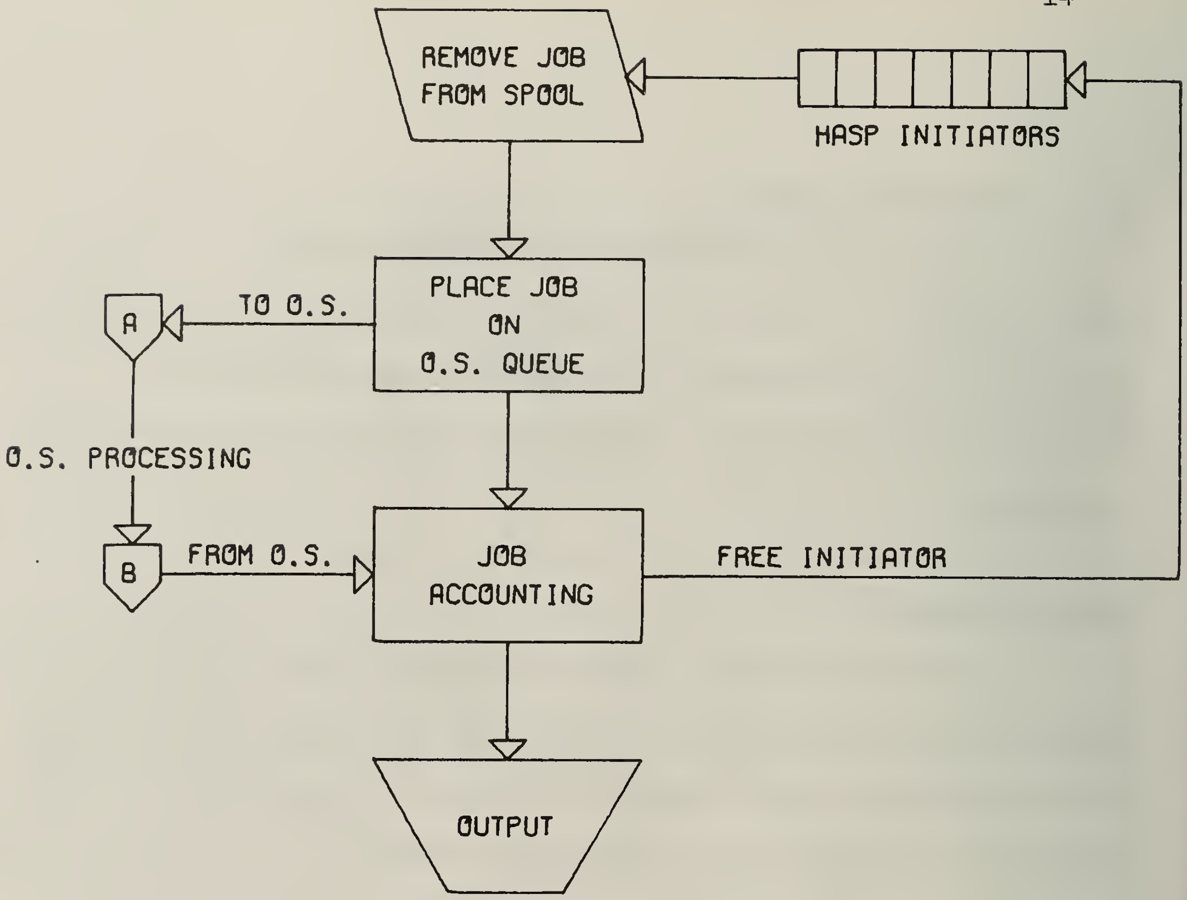


Figure 3. Logical Structure - HASP

O.S. Initiation

Corresponding to the HASP initiators are seven O.S. initiators, whose function is to take the job through the various stages of execution. The structure of O.S. is shown in Figure 4.

Preceding the execution of each step, the control cards are scanned for errors, and if no errors are detected the O.S. supervisor is called to allocate core. The first contiguous block of core large enough to contain the step request is allocated, with no attempt to compact used portions of memory to avoid fragmentation. If no space is available, the job must wait, hence tying up both its HASP and O.S. initiators.

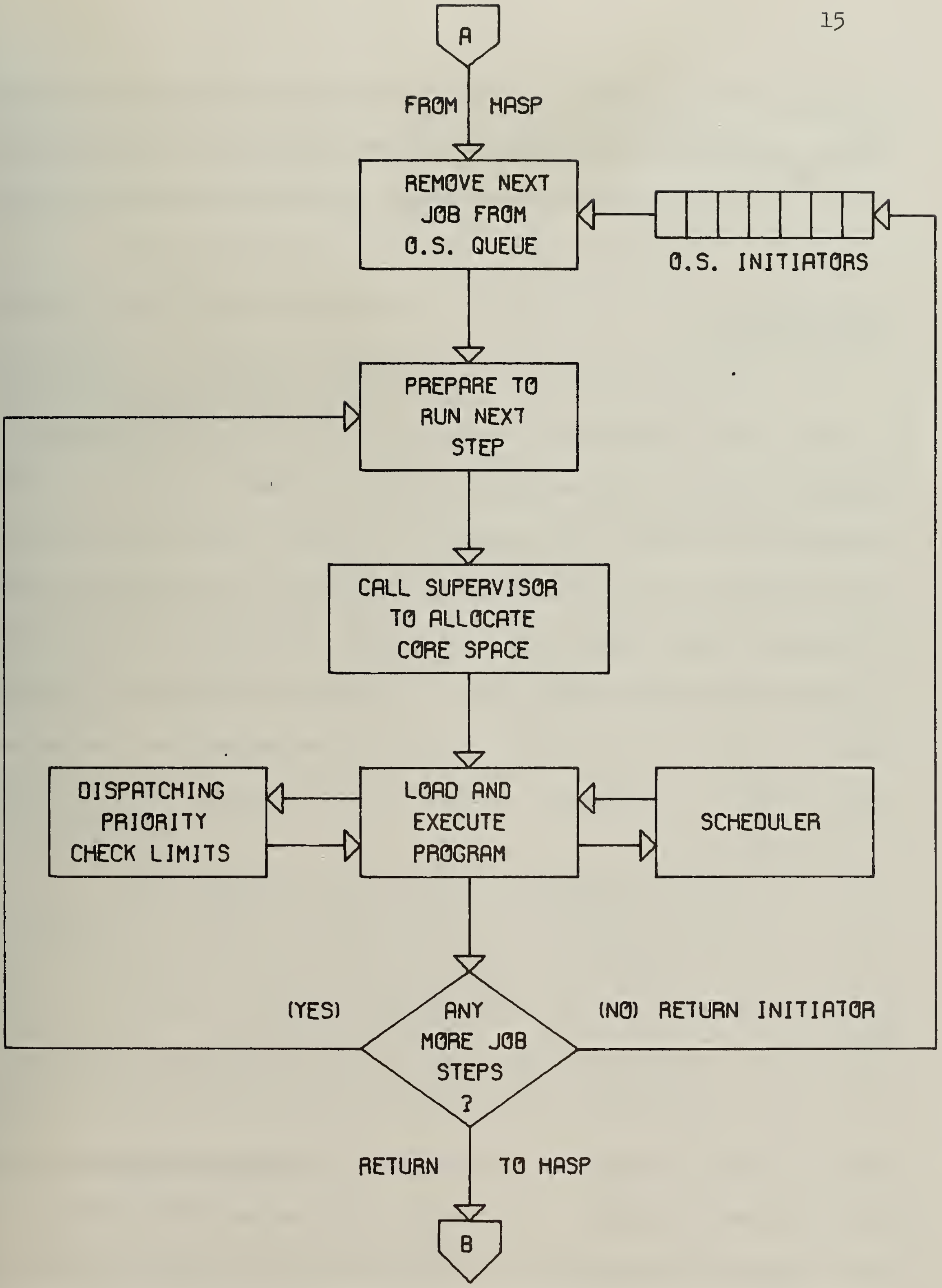


Figure 4. Logical Structure - O.S.

Following the allocation of core, data management is called to allocate devices, and the task is placed on the ready queue with the highest non-system priority to begin execution. The initiator waits for step completion.

HASP Dispatcher

During the execution of many tasks, precautions must be taken to ensure that a proper balance of system resources is maintained so that maximum utilization of facilities can be realized. If all of the jobs processing issue many I/O requests in relation to the number of CPU seconds used, the CPU would be under-utilized waiting for the requests to complete. On the other hand, if all jobs have a very high CPU-to-I/O ratio, the response time for short jobs would be very poor. Therefore, a well-balanced system consists of a majority of short jobs which use very few CPU seconds before issuing I/O requests, and one or two long jobs that will utilize any free CPU cycles.

The HASP dispatcher controls the interaction of the long and short jobs on the ready queue, and prevents either one from dominating the other. Every two seconds, the CPU is interrupted, and all jobs are returned to the ready queue. Based on resources used during the previous two second interval, a new dispatching priority is assigned to the task. Presently, the task with the lowest CPU-to-I/O ratio will get the highest dispatching priority, and will be the most likely to be assigned CPU cycles upon return to the operating system. This ensures that short tasks which need only a few cycles will be given the necessary time before issuing another

I/O request, and long CPU bound jobs will not be permitted to run until free CPU time is available. The interrupt every two seconds prevents the long jobs from completely taking over the system.

During the same period, statistics of resources used are updated, and if any job estimates are exceeded the job is immediately terminated.

O.S. Master Scheduler

The O.S. Master Scheduler is responsible for allocating CPU cycles to each of the tasks on the ready queue. As the current task using the CPU ends processing (due to a user or system interrupt) the scheduler will initialize the proper system status registers and give control of the CPU to the task with the highest dispatching priority assigned by the dispatcher.

Job Termination

When execution of all steps is complete, accounting records are updated and Print or Punch service is called to produce the actual output. The output service routines take over control of the job allowing the O.S. and HASP initiators to return to the system. Finally, purge service physically removes the job from the system.

3.2. Design Philosophy

Once the system to be modelled was defined, a design philosophy for the simulator had to be established. The model was intended to be a general purpose simulator, and hence was designed to be completely expandable. In addition, system functions were developed as independent modules so that any one function could be modified for study without affecting the operation of the rest of the system. Some of the design considerations are discussed in the following paragraphs.

Simulated Time Unit

A major decision regarding any simulation model is the length of the simulated time unit. A small time unit would be ideal for a computer system simulation. However, other ramifications of this unit must be considered. It is desirable to simulate a relatively long real-time period in order to study the effect of any system modifications. This would be extremely lengthy if too small a time unit were chosen, requiring an excessive amount of computer time. Also, depending on the level of simulation, the accuracy could actually deteriorate as a result of the fine division of time. These and other considerations led to the selection of 1 millisecond as the clock unit.

Using a time unit of 1 millisecond immediately results in the problem of accounting for times less than 1 ms. Of course, these times could not be ignored, but at the same time, could not be counted as a full clock unit. A compromise approach was used--that of accumulating all of these small pieces into a sum total of "system overhead", to be run during initation/termination.

It is impossible to measure accurately (within 10 percent) the amount of system time required for a given job. In a period of simulated time T, an error $E_s = T_s N_s e$ will be accumulated, where N_s is the number of relatively short (~1 ms) amounts of system time needed, T_s is the total time spent on each of these short intervals, and e is the percentage error in the simulated time given to this operation. Similarly, the error for long intervals E_ℓ can be shown to be $T_\ell N_\ell e$ where T_ℓ and N_ℓ are as above for some longer periods (~1000 ms). The simulation shows the ratio $T_s N_s / T_\ell N_\ell$ is approximately 2, resulting in a greater error with the smaller time unit.

System time chargeable to a job, therefore, is executed during initiation/termination. Additionally, nonchargeable overhead (accounting, scheduling, etc.) is accounted for at each interrupt of a job in the CPU, and at the reordering of dispatching priorities of jobs on the ready queue.

Entity Representation

Before proceeding to write a simulation program, careful consideration had to be given to the way in which actual system entities were to be represented by the simulator. The properties of a given system feature to be simulated had to be defined and the GPSS function most closely matching the requirements selected. For example, representation of the HASP Spools was of primary importance, and GPSS offers a number of possibilities--queues, chains, etc. The requirement that transactions be reordered at any time ruled out the queue representation, and the optional automatic priority ordering possible with a user chain led to its selection. Chains also offered the best method of communication between nodes of the network since it is possible to scan the chains and remove any specific job. This is essential for the implementation of any load leveling or system balancing algorithm.

The structure of GPSS played another important part in determining the representation of jobs. A job could have been represented as a table (in the literal programming and not GPSS sense) that would contain the information about this job, and be referenced by all other transactions in the simulation. This would have led to a simulation within the simulation, an undesirable effect. Therefore, jobs are represented as transactions which keep all pertinent information in parameters. Unfortunately, this led to some rather complex timing and communication considerations which had to be resolved before the simulator could run.

Timing and Communication

There is no direct method of communication between two transactions in GPSS, so whenever such contact was necessary, alternate procedures were devised. For example, at some points an initiator must know the size of core requested by a given job. The receiving transaction must put itself in a blocked state, while freeing the transaction from which the information is required. The information is then put in a savevalue or other temporary location by the sending transaction. After signalling the receiving transaction that the information is present, this transaction puts itself in a blocked state, and thus allows the receiving transaction to regain control of the simulator in order to pick up the contents of the savevalue. This procedure is non-trivial, since in an event-driven simulation, there may be any number of transactions ready to run when the sending transaction is blocked. The priorities of the ready transactions, and knowledge of the scheduling algorithms of the simulation language itself, must be analyzed to ensure correct results.

During the simulation, the jobs waiting to be executed are not the only transactions waiting to use the simulated CPU. Transactions representing the scheduler and dispatcher also require this facility. Therefore, it is required that only one transaction enter the CPU at any given time since this is not a multiprocessing environment. Logic switches are set and facility usage tested by every transaction requesting the CPU.

3.3. GPSS Implementation

With this design philosophy, I will proceed to outline the representation of the various HASP and O.S. entities at each node. The logical structure of the simulation processes occurring at each node, shown in the flow charts of Figures 5 through 8, are summarized in the following paragraphs [7], and a basic knowledge of GPSS is assumed.

HASP and O.S. Initiators (Figure 5)

Two sets of initiators are needed, one for HASP, and one for O.S., each requiring the same information about the jobs they are servicing. In addition, the HASP initiator for a specific job must be dormant while the O.S. initiator is running. Therefore, seven transactions are created, each of which represents either a HASP or O.S. initiator. Each transaction is created as a HASP initiator and put in an inactive state awaiting the arrival of jobs. After the initiator completes initiation of a job and places it on the O.S. queue, the HASP initiator becomes the O.S. initiator. This O.S. initiator flows through the core allocation and other resource allocation routines to request core space, and finally places the job on the ready queue to run. This initiator then becomes dormant waiting for the job (or job step) to complete. At each step completion, the initiator is awakened to request resources for the succeeding step. When the entire job completes, the initiator is returned to an inactive state where it again performs its HASP function.

Whenever an initiator or job is to be put in an inactive state, it must be taken off the current events chain and placed on a chain specifically representing that wait condition.

Jobs (Figure 6)

Each job is represented by one GPSS transaction with parameters containing information such as creation time, number of milliseconds that will be executed, size of core requested, etc. The parameters are referenced throughout the simulation to keep a record of what was done, and indicate what the next step will be. In this way, by moving the transaction from one section of the model to another, different stages of execution can be indicated.

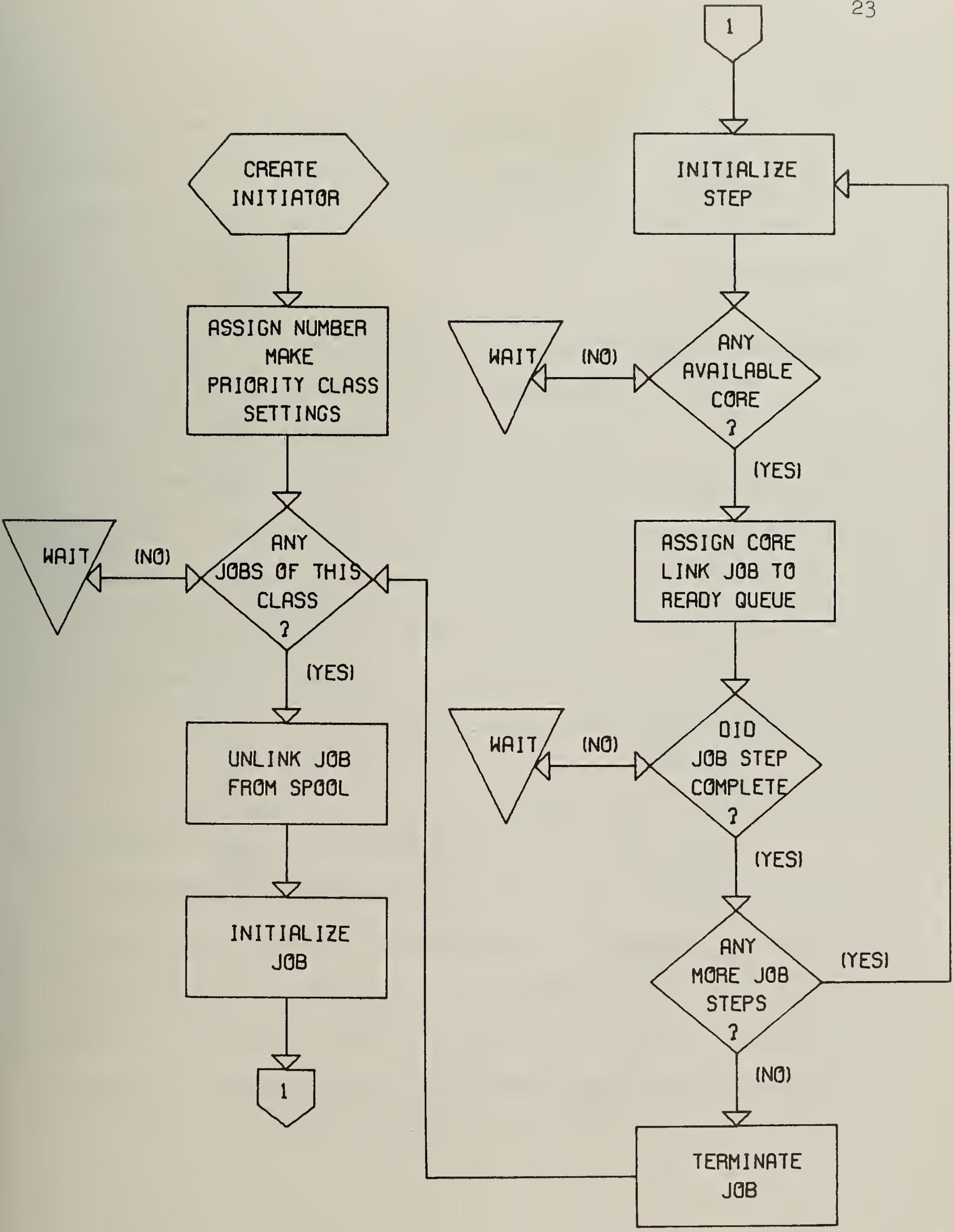


Figure 5. HASP and O.S. Initiation

Queues

All HASP and O.S. queues are represented by user chains as discussed earlier. In addition to facilitating ordering of objects on the queues, chains gather the proper waiting time and size statistics automatically.

Dispatching Priority Assignment (Figure 7)

The HASP dispatching priority assignment is carried out by one final transaction. Every 2 seconds this transaction is given control of the simulator, and proceeds to reassign the dispatching priority of the jobs on the ready queue and those jobs currently issuing I/O requests. The job in control of the CPU (if any) is interrupted, and placed back on the ready queue according to its new priority.

When all of the reordering is complete, the scheduler is freed, and the dispatcher is made dormant for another two seconds.

CPU - Scheduler (Figures 8a and 8b)

The scheduler has the responsibility of determining which task will next get control of the CPU. The scheduler is represented by one high priority transaction that unlinks jobs from the ready queue and lets them seize the facility corresponding to the CPU. While this job is advancing the clock in the facility, no other transactions are permitted to enter. Although GPSS automatically permits only one job in each facility, this is not sufficient protection against more than one

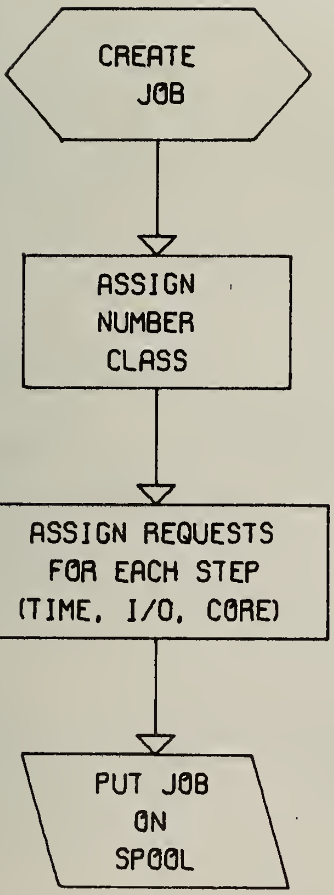


Figure 6. Job Creation

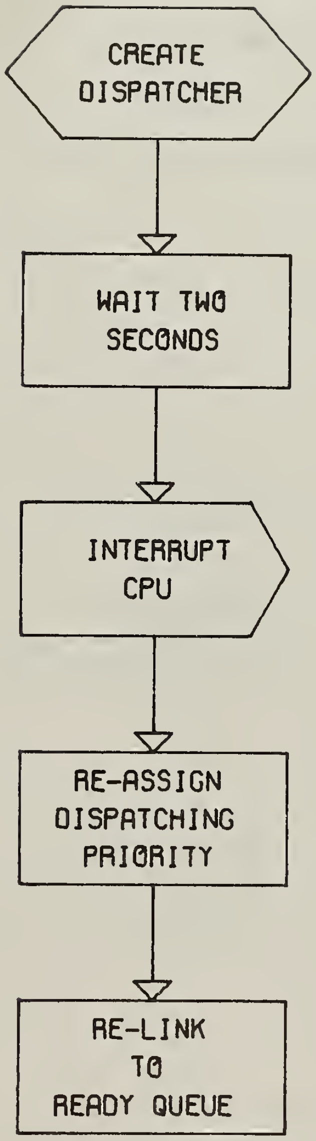


Figure 7. HASP Dispatcher

transaction entering the CPU. Therefore, this condition is explicitly tested by all transactions requesting the CPU. Multiple transactions are allowed to enter blocks representing I/O requests, and other system processes, since these functions in the real system are actually carried out in parallel. When the CPU is released, control is returned to the scheduler, which allocates the facility to the next job on the ready queue.

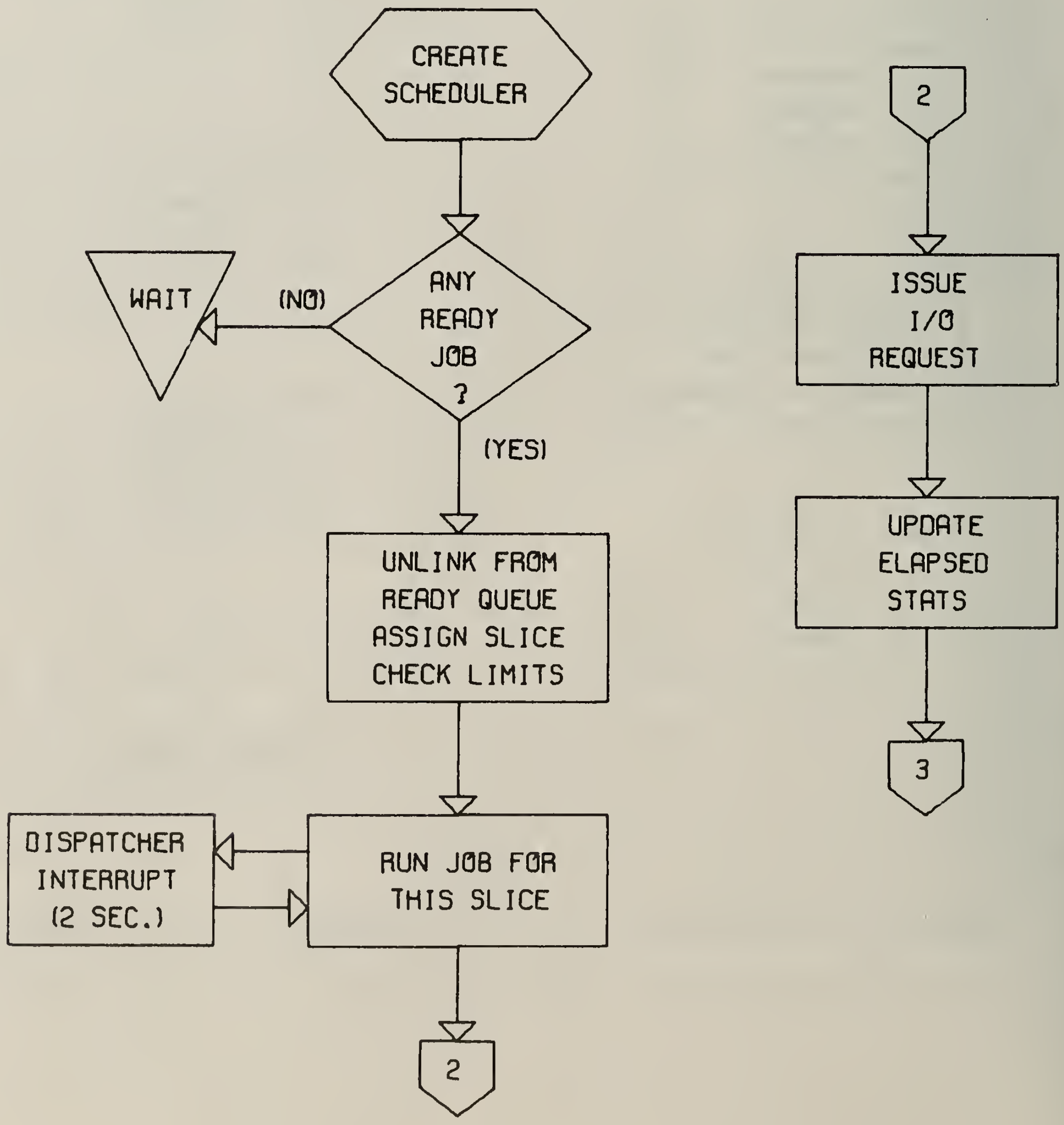


Figure 8a. Central Processing Unit

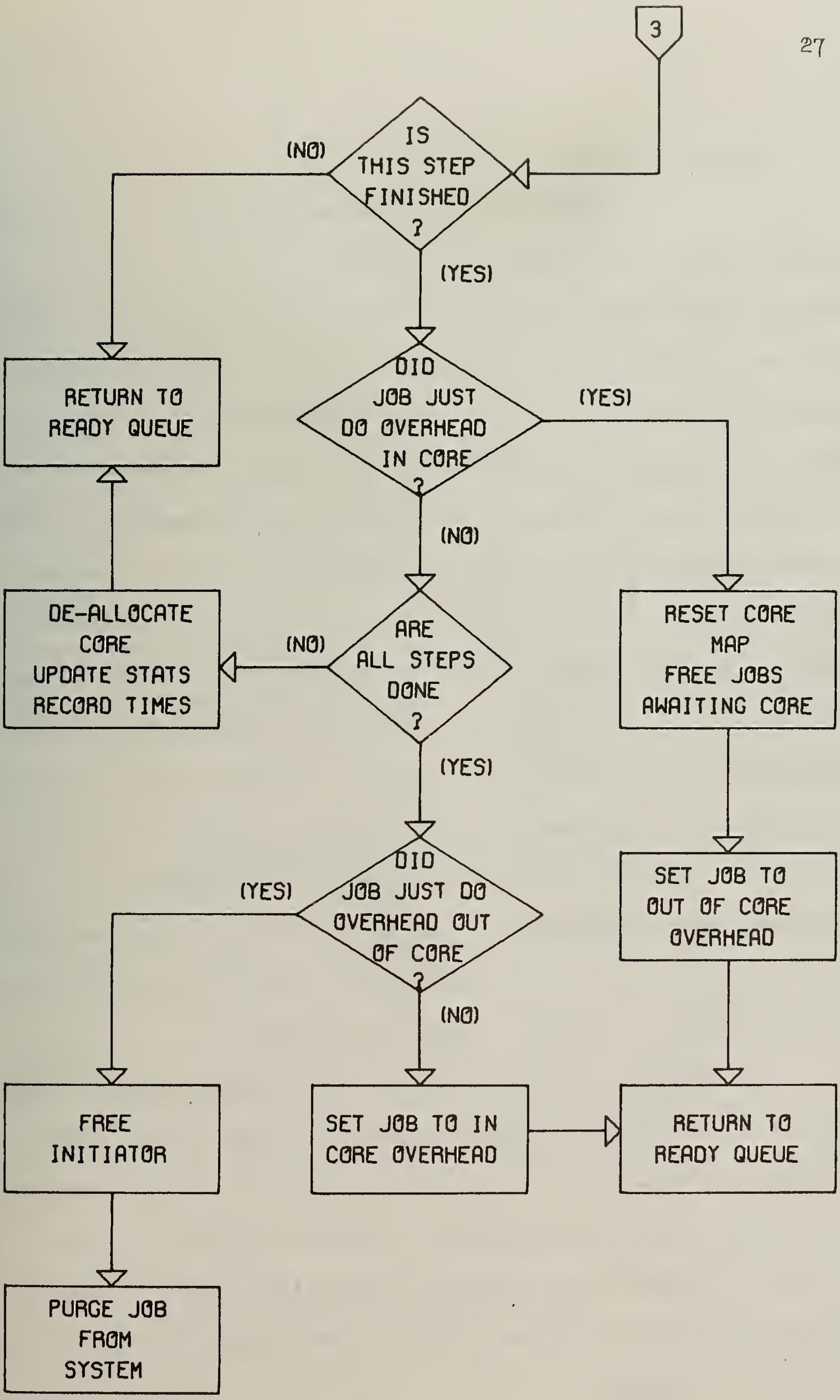

3
IS THIS STEP FINISHED ?
(NO)
(YES)
RETURN TO READY QUEUE
DID JOB JUST DO OVERHEAD IN CORE ?
(YES)
(NO)
DE-ALLOCATE CORE UPDATE STATS RECORD TIMES
ARE ALL STEPS DONE ?
(NO)
RESET CORE MAP FREE JOBS AWAITING CORE
(YES)
DID JOB JUST DO OVERHEAD OUT OF CORE ?
(YES)
SET JOB TO OUT OF CORE OVERHEAD
(NO)
FREE INITIATOR
SET JOB TO IN CORE OVERHEAD
RETURN TO READY QUEUE
PURGE JOB FROM SYSTEM

Figure 8b. Central Processing Unit

3.4. Results Obtainable

At the same time the model was being developed, the information that was to be collected had to be determined. Since sections of the simulator were written explicitly in GPSS, as opposed to using built in functions of other simulators, any desired results could be obtained. The goal of turnaround time for users prompted a heavy emphasis to be placed on the time at which steps and jobs completed. These times were tabulated so that general conclusions could be reached. Unfortunately, the bare statistics can be quite misleading, and must be interpreted carefully. Table I shows tabulated values for the time simulated jobs waited on the HASP queues. On the surface, the table seems to imply that jobs had to wait longer to be processed on the network than by independent centers without load-leveling. However, closer inspection will reveal that approximately 30 percent more jobs were processed by the network, leaving less than half as many jobs remaining on the queues. Calculating the total number of seconds waited by all jobs, and dividing by the number of jobs in the three systems, yields the average wait time per job, as shown in Table II, which are as expected.

In addition, these statistics can be broken down into separate priority classes for more detailed study. Although overall system statistics may appear to be stable from one case to another, some very serious changes can be taking place within the individual classes. Therefore, all figures on turnaround time, waiting time, etc., are collected for each of the five priority clsses at each center as well as the network as a whole.

TABLE I

Sample Simulated Waiting Times

	WITHOUT LOAD LEVELING		WITH LOAD LEVELING	
TIME ON SPOOL (Jobs Processed from)	Time[1]	No. of Jobs	Time	No. of Jobs
Center 1	43	120	51	197
Center 2	127	57	156	50
Center 3	0	28	42	20
Network	60	205	69	256
TIME ON SPOOL (Jobs Still Waiting at)				
Center 1	593	182	616	69
Center 2	618	16	595	23
Center 3	32	1	380	9
Network	592	199	590	101

[1]Time in Seconds

TABLE II

Normalized Simulated Waiting Times

AVERAGE TIME ON SPOOL	Time[1]	Time
Center 1	374	197
Center 2	234	294
Center 3	1	146
Network	327	212

[1]Time in Seconds

Other features of GPSS, however, produce useful statistics in a more straightforward manner. Since "Facility" entities gather utilization figures as part of their internal function, the required statistics were collected merely by having the transaction seize the appropriate facility. For example, the simulated CPU's are represented by five facilities in each center, each corresponding to use by the problem program, system overhead (two phases), dispatcher overhead, and scheduler overhead, respectively. The percentage of utilization by each of the above functions is easily observed, and the total utilization of the system is obtained from the sum of these values.

Core utilization figures are obtained by using the GPSS "storage" entity. A storage is essentially the same as a facility, except that more than one transaction may seize it at one time. One element of storage corresponds to one kilobyte of simulated core, and transactions requested core seize the number of elements corresponding to the request.* Figures for the storage capacity (total core available), average number of kilobytes used, average utilization, total number of kilobytes used, and the average time the elements were in use are directly available. Figure 9 shows a sample of the information directly produced by the simulator, with a complete discussion of the results to be found in Section 4.

*The assignment of core to a job is carried out through the use of a core map, which allows core contention and fragmentation to occur. Only after the region is assigned in this manner is the storage entered for statistical purposes.

```
FACILITY          AVERAGE          NUMBER          AVERAGE
                UTILIZATION        ENTRIES        TIME/TRAN
       1           .167            138810           22.898
       2           .015              3323           87.383
       3           .015              3204           93.056
       4           .024              9263           50.000
       5           .198            143492           26.266
```

Figure 9. Sample Simulator Output

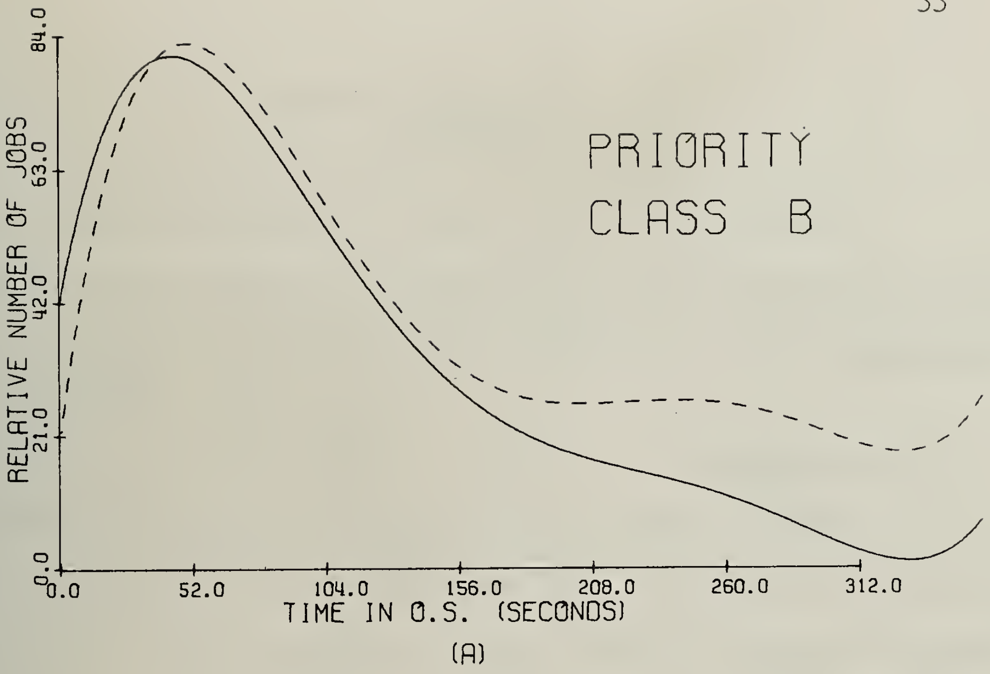
3.5. Validation

Since a simulation study typically models nonexistant situations, comparisons of the simulated results with a known quantity is quite difficult. However, if the model is developed in small stages, each stage can be compared to some existing system. By combining the individual modules, each of which has been verified accurate, the complete model will maintain a high degree of accuracy. In this study, each node of the network was tested by comparing simulated results to the 360/75 system current in operation at the University of Illinois at Urbana-Champaign.

The process of comparing the model and the real system consisted of two steps. The first task was to adjust the parameters of the model so that they corresponded with the 360. Specifically, initiator settings, dispatcher interrupt intervals, and other system parameters had to be correct before any comparisons could be made. At the same time, these settings were investigated, information was gathered on the jobs arriving at the system so that distribution functions could be created and used by the simulator. Statistics for CPU seconds used, number of input/output requests issued, core region size requested, and number of steps executed were gathered. This information was converted into GPSS functions and used to create the simulated job stream.

The second task of the validation was to compare the performance of the two systems. From the same records that supplied the resource utilization statistics used as input to the simulator, figures on the real time these jobs spent in the system were also collected. With this information, it was possible to run the simulator with the input statistics, and predict the time that these jobs would spend in the system. The prediction of the simulator for one priority class, and the actual time as gathered by system accounting records, were graphed and can be seen in Figure 10a. The graph for all of the jobs that ran at one center is shown in Figure 10b.

One indication that the simulator yields accurate results is seen by observing that although the shapes of the graphs of one priority class differ from those of another, the simulated and actual system graphs are similar in shape. Differences in the graphs were resolved by adjusting parameters in the simulator until the desired degree in similarity was obtained. This process is being continued using more complete actual system data obtained from controlled "benchmark" runs on the 360/75.



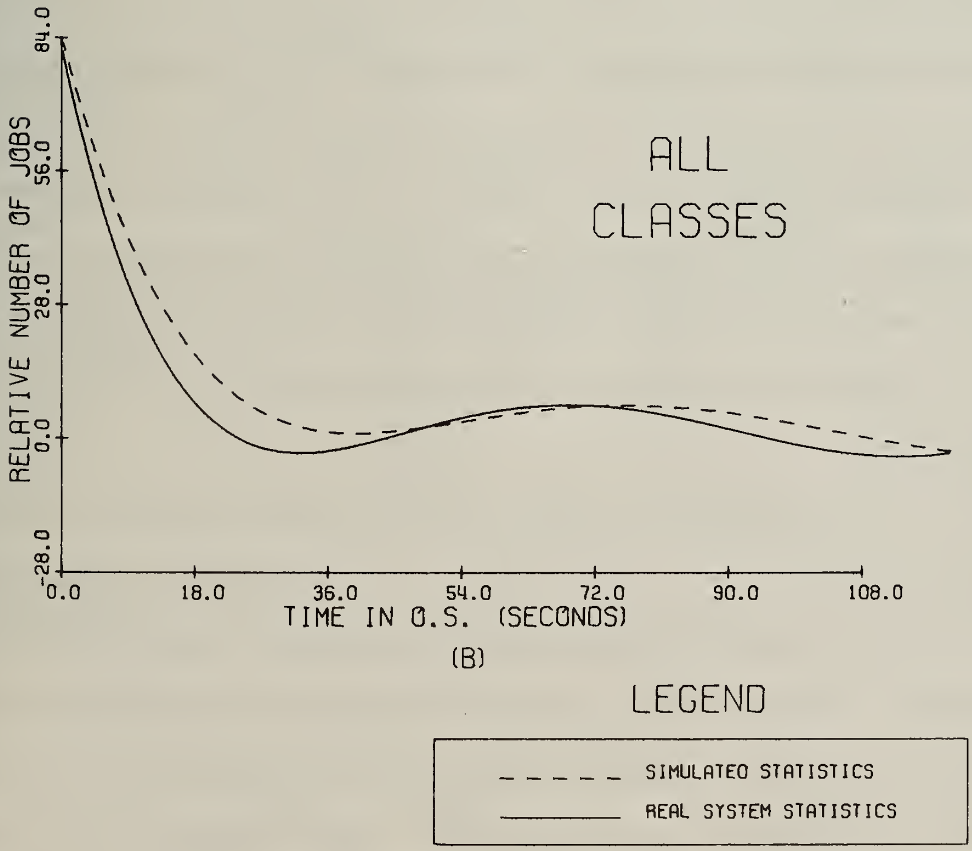


Figure 10. Comparison of Real and Simulated Statistics

4. SIMULATION RESULTS

Once the model had been shown to produce an accurate picture of the system being studied, the proposed algorithms were simulated and analyzed.

4.1. Load Leveling

The first algorithm to be tested was the load leveling process as discussed in Section 2. To demonstrate the use of the model, the three node network of Figure 2 was constructed so that centers had an interarrival rate of $\lambda_1=5$ seconds, $\lambda_2=15$ seconds, and $\lambda_3=30$ seconds. This produced a network where the demand on center 1 was very heavy, center 2 was not quite as heavy, and center 3 was very light. In order to adequately study the effects on such a configuration, separate simulation runs were necessary to demonstrate the consistency of the results under varying conditions. Table III shows queue waiting time statistics for one simulation run, which clearly demonstrates the effects of the algorithm.

As expected, the effect of the change is different at each of the individual centers. For center 1, the most heavily loaded center, the time spent waiting on the queue has decreased significantly with load leveling. Jobs at centers 2 and 3, however, have suffered an increase in waiting time, and the question arises as to whether or not the new system is beneficial for these centers. The inspection of the remaining data in Table IV sheds light on the answer. The number of jobs processed

TABLE III

Waiting Times Comparisons in a Network Computer*

	WITHOUT LOAD LEVELING		WITH LOAD LEVELING	
TIME ON SPOOL (JOBS PROCESSED)	Time**	No. of Jobs	Time	No. of Jobs
Center 1	73	195	85	273
Center 2	121	80	93	87
Center 3	0	47	101	39
Network	74	322	88	399
TIME ON SPOOL (JOBS NOT PROCESSED)				
Center 1	1155	107	821	29
Center 2	442	16	909	9
Center 3	0	0	385	8
Network	1062	123	762	46
AVERAGE TIME ON SPOOL				
Center 1	456	302	155	302
Center 2	175	96	169	96
Center 3	0	47	149	47
Network	347	445	157	445
JOBS PROCESSED (FROM CENTERS	No. of Jobs		No. of Jobs	
Center 1	189		262	
Center 2	74		85	
Center 3	45		37	
Network	308		384	

*40 Minutes Simulated Time
**Time in Seconds

TABLE IV

Resource Utilization in a Network Computer*

LOAD LEVELING	WITHOUT	WITH
CPU UTILIZATION		
Center 1	.961	.958
Center 2	.911	.895
Center 3	.624	.884
Network	.832	.912
CORE UTILIZATION		
Center 1	.753	.762
Center 2	.736	.732
Center 3	.311	.726
Network	.600	.740
JOBS PROCESSED (AT CENTER)		
Center 1	189	147
Center 2	74	115
Center 3	45	119
Network (Avg/Center	102	127
REVENUE COLLECTED		
Center 1	$ 658	$ 537
Center 2	300	428
Center 3	190	446
Network	1148	1411
*40 Minutes Simulated Time		

at the centers increases (increased throughout), both the CPU and core utilization figures increase (increased resource utilization), and the revenue collected rises sharply. The trade-offs between increased turnaround time at centers 2 and 3, and greater resource utilization and throughput at these centers must be considered by the computing center manager. The statistics for the network as a whole indicate the advantages of load leveling for this configuration.

4.2. Pay-for-Priority

As a further demonstration of simulation techniques used in this study, a basic pay-for-priority scheme was implemented. As each job enters the model, a reward factor r_i, where $1.0 < r_i < n$ is assigned, and n is determined by the analyst. This reward is then used to determine the job's priority, P_i, is given by

$$P_i = \frac{PRT}{(r_i)^2} ,$$

where PRT is the intial static priority based on user estimates of resource use. When processing completes, the charges C_i for the job are computed by

$$C_i = c_i r_i ,$$

where c_i are the nonweighted charges for the resources used.*

The theory of pay-for-priority seems straightforward enough to expect that more revenue will be collected using this scheme, even though the turnaround time for low priority jobs may increase. However, Table V shows that other factors must be considered to avoid total degredation

*The processing charges c_i at the University of Illinois at Urbana-Champaign are currently computed by

$$c_i = 0.04(X+Y)(0.0045Z+0.5)$$

where X = CPU time in centiseconds,
Y = number of Input/Output requests, and
Z = kilobytes of core used.

of performance. Using λ_1=2.5 seconds, λ_2=10 seconds, and λ_3=20 seconds results in a drop in CPU utilization, a decrease in throughput, and at the heaviest loaded center, an actual loss of income.

TABLE V

Evaluation of Pay-for-Priority Algorithm*

PAY-FOR-PRIORITY	WITHOUT	WITH
REVENUE COLLECTED		
Center 1	879	816
Center 2	852	852
Center 3	888	963
Network	2592	2634
CPU UTILIZATION		
Center 1	.977	.964
Center 2	.922	.948
Center 3	.887	.873
Network	.935	.928
CORE UTILIZATION		
Center 1	.761	.763
Center 2	.716	.717
Center 3	.677	.722
Network	.718	.734
NO. OF JOBS PROCESSED AT THESE CENTERS		
Center 1	264	228
Center 2	216	225
Center 3	237	225
Network	717	678

*One hour Simulated Time

As jobs increase their priority, the optimality of the scheduling algorithm is violated resulting in a decrease in the number of jobs processed. Under these conditions, the loss of income from nonprocessed jobs is greater than the gain from those jobs paying for increased priority. At center 2, however, the scheme seems to have very little effect on overall performance of the center, indicating it is at some "break-even" point. Although fewer jobs were processed, the difference was equal to the gain from jobs processed at the higher rate. When the demand on the center drops below this point, the change in priority will not effect the throughput of the system, and, therefore, a rise in income will be a result of the priority changes. This phenomenon can be seen in the statistics for center 3 in Table V.

The simulation results, therefore, indicate that the job charges should be computed as

$$C_i = c_i[(r_i-1)h_j+1]$$

where C_i = total charges for this job,

c_i = nonweighted charges for this job,

r_i = reward factor specified by the user, and

h_j = a constant multiplicative factor determined by the load on system j. As L_j increases, h_j increases to ensure that the center receives sufficient compensation for disturbing the scheduling process under varying conditions. Table VI shows the results of the same network (with identical job streams) with $h_1=2.0$, $h_2=1.5$, and $h_3=1.0$. Clearly, by setting $h_1 > 2.0$, center 1 would experience an increase of income proportional to the gains of centers 2 and 3.

TABLE VI

Adjusted Pay-for-Priority Algorithm

Pay-for-Priority	Without	With
REVENUE COLLECTED*		
Center 1	$ 879	$ 936
Center 2	852	933
Center 3	888	963
Network	2592	2832

*One Hour Simulated Time

CONCLUSION

As computing machinery becomes increasingly complex, techniques for sharing resources must be developed. With an increase in the number of computing applications and users, algorithms must be implemented which are designed to distribute the load of computing centers and increase the performance of the computer equipment.

This paper presents scheduling algorithms and load leveling techniques for a network of computers. Criteria for performance evaluation are defined, and the algorithms are implemented on a network simulator. The techniques are then analyzed in terms of these performance criteria.

A pay-for-priority scheme, allowing the user to receive better service for a higher price, is studied. It is shown that a simple scheme allowing users to increase their priority disturbs the optimum nature of the standard scheduling algorithms, and at a heavily loaded center, a complete degredation of performance occurs. Therefore, the load on the system is to be an important factor in determining the cost for running a job at a higher priority.

These algorithms are then implemented with load leveling and communication between centers, and the dramatic increase in system performance for the centers is shown through the simulation. Although a slight increase in turnaround time results at the lighter centers due to jobs being processed from the heaviest centers, the resource utilization figures and revenue collected sharply increase.

This study demonstrates the desirability of further investigation of network computers. Furthermore, the simulation shows that benefits can be obtained by considering that jobs are not of equal value, and scheduling their execution accordingly. Investigation in both of these areas is continuing, and the simulation model is being updated and improved for further study.

With the improved model, it remains to further investigate other scheduling algorithms and priority techniques. An obvious next step is a detailed simulation study of the cost effective priority assignment algorithm discussed in Section 2.3., and its effect on network performance. This study should result in the development of techniques necessary to take full advantage of the benefits offered by network computers.

LIST OF REFERENCES

[1] Barr, W. J., "Cost Effective Analysis of Network Computers," Department of Computer Science Report No. UIUCDCS-R-72-538, University of Illinois at Urbana-Champaign, August 1972.

[2] Bowdon, E. K., Sr. and Barr, W. J., "Cost Effective Priority Assignment in Network Computers," Proceedings of the Fall Joint Computer Conference, December 1972.

[3] Bowdon, E. K., Sr., Mamrak, S., and Salz, F., Simulation: "A Tool for Performance Evaluation in Network Computers," Proceedings of the National Computer Conference and Exposition, June 1973.

[4] ______________, General Purpose Simulation System/360 Introductory User's Manual, GH20-0304-4, International Business Machines Corporation, 1968.

[5] Mamrak, S., "Proposed Priority Scheme for the IBM 360/75," Non-circulated internal document, Department of Computer Science, University of Illinois at Urbana-Champaign, February 1973.

[6] Rutledge, R. M. et. al., "An Interactive Network of Time-Sharing Computers," Proceedings of the 24th National Conference Association for Computer Machinery, 1969.

[7] Salz, F., "A GPSS Simulation of the 360/75 Under HASP and O.S. 360," Department of Computer Science Report No. UIUCDCS-R-72-528, University of Illinois at Urbana-Champaign, June 1972.

[8] Schrage, L., "A Proof of the Optimality of the Shortest Remaining Processing Time Discipline," Operations Research, Vol. 16, 1968.

[9] Simpson, T. H., Houston Automatic Spooling Priority System II, International Business Machines Corporation, 1969.

IBLIOGRAPHIC DATA HEET	1. Report No. UIUCDCS-R-73-578	2.	3. Recipient's Accession No.

Title and Subtitle

Simulation Analysis of a Network Computer

5. Report Date May 18, 1973

6.

Author(s) Fredrick R. Salz

8. Performing Organization Rept. No. UIUCDCS-R-73-578

Performing Organization Name and Address
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

10. Project/Task/Work Unit No.

11. Contract/Grant No. NSF GJ 28289

Sponsoring Organization Name and Address
National Science Foundation
1800 G Street, N.W.
Washington, D.C. 20550

13. Type of Report & Period Covered
Master of Science Thesis

14.

Supplementary Notes

Abstracts

As computing needs increase, the resources at a given installation become nadequate to satisfy the requirements of all users of the facility. Presently, hen the deficiency of equipment occurs, the feasibility of purchasing additional esources is investigated, and if sufficient funds exist new units are added to the ystem. A network computer offers virtually an unlimited capability for expanding omputer resources without the costs involved in having all of the equipment at the ndividual installation. Since the needs of computing centers vary with time, idle esources normally existing during a light load period can be utilized by another verloaded center. What is presented here is an investigation of the benefits btainable through the use of networking. A "pay-for-priority" scheduling algorithm r the individual centers is presented, and load leveling techniques for transmitting bs between centers is discussed. The concepts are then demonstrated through the use a simulation model for a general network computer. Finally, the effects of these gorithms on system performance are demonstrated through the use of a simulation model for a general network computer.

Key Words and Document Analysis. 17a. Descriptors

mulation
stems Analysis
twork Computer Analysis
stem Modeling

Identifiers/Open-Ended Terms

COSATI Field/Group

ailability Statement Release Unlimited	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 46
	20. Security Class (This Page UNCLASSIFIED	22. Price -----